TD-09-024

Optical inspection of SRF cavities

Evgeny Toropov,
Moscow Institute of Physics and Technology
TD/SRF

Ernesto Tripodi
Università degli Studi di Pisa
TD/SRF

Optical inspection of SRF cavities

The general task

1) Cavity
2) Problems
3) Software to develop

Hardware

1) Components
2) Characteristics
3) Limitations
4) Interface & Software developed earlier

Software for motors operating

1) Software developed earlier
2) Necessity for separate engines
3) Motor engines SubVIs
4) Modification of engines
5) Operations description

Autofocus software

1) Necessity and requirements
2) Focus function
3) Choice of function and parameters
4) Approaching focused image
5) Autofocus software

Software for cavity shape analysis

1) Sample measurements. Results
2) Processing results
3) Cavity shape
4) Errors and limitations

Software for image collecting

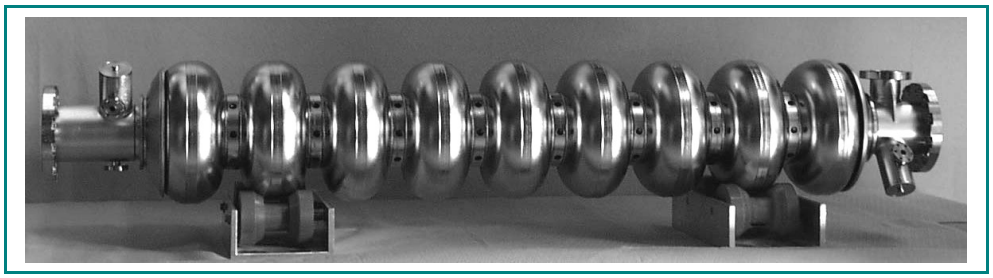
1) Algorithm
2) Interpolation

Conclusion

The general task

Cavity

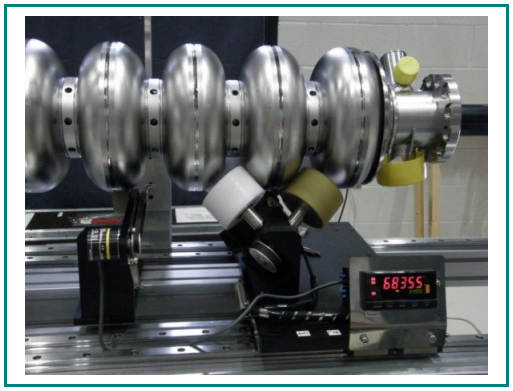
SRF-cavity is a 1-meter 9 cell resonator, where altering electric field can be used for accelerating particles (electrons and positrons in linear accelerators). A bunch of particles passes through each of nine cells consequently. While it passes through one of the cells the electric field accelerates it. The frequency of the field is adjusted so that the direction of the field in this particular cell switches to the opposite only when the bunch is gone. At the same time, the field in the next cell starts accelerating the particles.

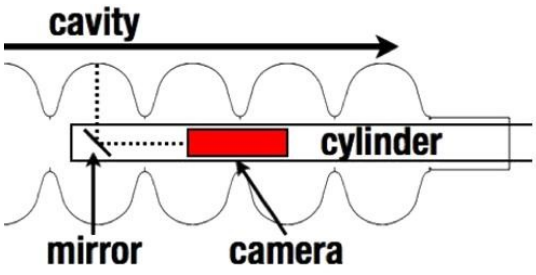


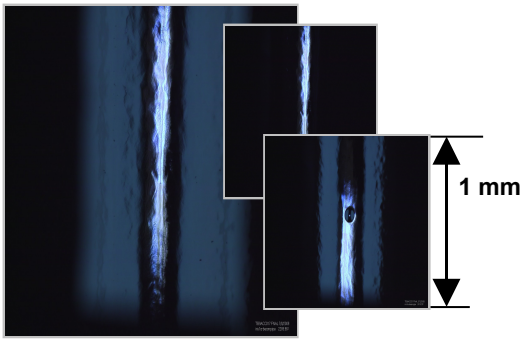
To enhance the performance of cavity acceleration, cryogenic temperature conditions were employed.

Problems

However, the technology of cavity production is not perfect and various defects can be found occasionally in the inner side of inter-cell and equator welds. For different reasons these defects can cause serious downfall in performance. Hence, before the costly tests each cavity undergoes inspection for these defects. One of the methods is an optical inspection.







During the optical inspection a camera is placed inside the cavity. The snapshots can be used for tracing out these defects. Therefore, to find all the defects it is necessary to inspect the entire inner surface using this camera. The number of snapshots required is several thousands, what makes the process difficult to perform manually in case the number of cavities inspected is large. Automation of the inspection was the task of our internship in the summer of 2010.

Software to develop

During the internship we developed software for automation of optical inspection. It was required to solve several problems:

1) auto-focus on the necessary part of the image
2) collecting images of the entire surface
3) cavity shape imprecision study

LabView and MatLab software proved to be most suitable for completing these tasks. LabView was chosen for two main reasons: it is widely used in the Laboratory for mechanical engineering and it is very convenient for working with various hardware. MatLab was also extremely useful since it allows performing research using advance mathematical algorithms libraries.

Hardware

Components

1. 'Table motor' adjusts the position of the camera inside the cavity

   1) 1st control 'Shift'

   shifts the cylinder with the camera forward and backwards inside the cavity

   2) 2nd control 'Rotate'

   rotates the cavity

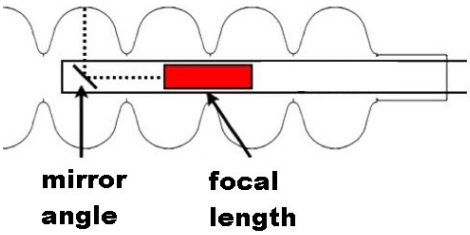
2. 'Camera motor' adjusts the camera properties

   1) 1st control 'Focus'

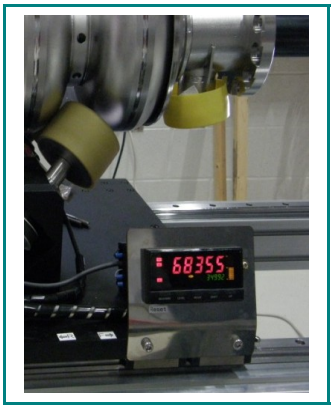
   changes the focal length of the camera

   2) 2nd control 'Angle'

   changes the angle of the mirror



3. 'Omron' indicator measures the angle of the cavity more precisely than 'Table motor' -> 'Rotate'



4. 'Camera' receives the image from the camera. The brightness of the image depends on the brightness of LCD and Side light.
5. LCD and Side lamps light the cavity from inside. Their brightness is regulated manually.

LCD lamp lights the central bright stripe. Side lamps bring light to areas outside the central stripe.

Characteristics

1. 'Table motor'

   1) 'Shift' (x-coordinate)

   Steps : 1 cm = 652 ± 3 steps (on speed 4) due to shift effect

   Speed : from 1 to 6. We have been using speed 4

   2) 'Rotate' (y-coordinate)

   Steps : 360° = 11120 ± 25 steps (on speed 3) due to shift effect

   Speed : from 1 to 6. We have been using speed 3

2. 'Camera motor'

   1) 'Focus'

   Steps : 1 mm = 270 steps ± 5 (on speed 3) due to shift effect

   Speed : from 1 to 6. We have been using speed 1-3

3. 'Omron'

   Steps : 360° = 11707 ± 2 steps

   Direction opposite to 'Table motor' > 'Rotate'

   Min = -20000 steps

   Max = 100000 steps

   Minimum and maximum values make us return to zero angle position after every full circle

4. 'Camera'

   1400 × 1000 pixels

   Resolution – 4 pixels × 3 pixels

   Color image

Limitations

1. 'Table motor'

   Shift-effect in table control is the main limitation. It is the hardware effect which cannot be eliminated. The image part of interest is a stripe which goes in y-dimension what makes it important to adjust the y-position (cavity angle)

precisely. At the same time, x-position (forward-backwards position) is not necessary to be adjusted because a user (as well as computer) can always determine the borders of the stripe.

'Omron' indicator is used for *adjusting the angle position*.

1) 'Shift' control

   On speed 4 shift-effect is not important

2) 'Rotate' control

   On speed 3 shift-effect is 25 steps per full turn while 3-5 steps shift is acceptable.

2. 'Camera motor'

   1) 'Focus'

   Data from 'Camera motor' > 'Focus' is used to perform *auto-focusing*. We consider shift-effect which appears when the motor changes direction to be partly responsible for imperfections in auto-focus. Namely, images from the camera differ for the same value of 'Focus' motor position. Therefore, we used additional *mechanism* to eliminate this effect.

3. 'Omron'

   Shift-effect is 2-3 steps for full circle which roughly corresponds to 2-3 steps of 'Rotate' motor.

4. 'Camera'

   It takes some time for camera to take a picture. Therefore, we allowed for some delay after using motor before grabbing snapshot. Tests show that this time must be at least 0.4-0.6 sec.

Software for motors operating

Interface & Software developed before

The programs use serial port interface to communicate with hardware. All the communication can be implemented with the use of LabView.

The software written before combined this low level communication into more complicated VIs (functions). There are VIs for both motors and 'Omron' indicator.

Omron:

There was a special sub-function developed for Omron indicator. On every call it initializes Omron, reads data and finishes the session.

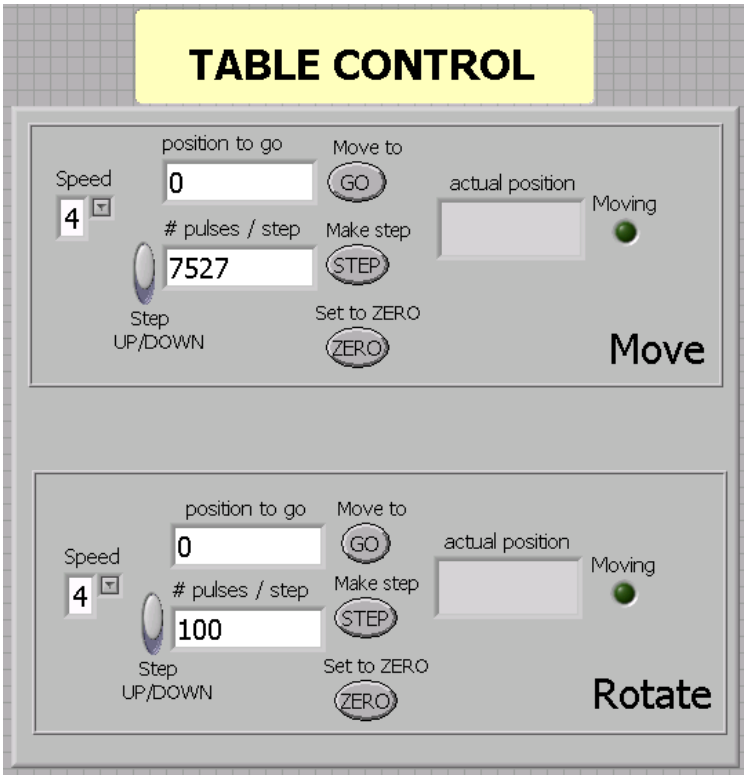
The only input is COM port. The output is position number.

Motors:

Motors use the same serial port communication interface and therefore the only difference from Omron is the COM port number. We can write data to serial port and read the Actual Position and Working motor state.

The program written before first initializes the motors, then it runs a loop continuously. After the stop command it ends the session. On every iteration the program checks if any action is launched (if a button is pressed). There are 3 possible actions for each of 2 controls of a motor:

1) go to position X ('go to')
2) make X steps ('steps')
3) set Actual Position to zero ('set to zero')



If any of these actions for any of two motor controls is launched, the program writes the necessary information to the serial port, reads the Actual Position and motor status (Moving?). If no action is launched the program only reads information.

For security reason the default value of Position To Go input was set as '?IDN'. Therefore, in your code make sure that you set this variable to any value when using this function. Otherwise the program waits infinitely and finally creates an error.

In all of the software described below these programs were used as sub-functions.

We wrote a program which combined these VIs for two motors and takes the image from the camera. It allows to manually control all the hardware from the computer. (*Control.vi*)

Necessity for separate engines

In software that can perform auto-focusing or changing the angles automatically each of motor operations creates certain difficulties.

First, if we want to use the programs described above as sub-functions every time we execute an operation we have to first launch an action, then wait until Moving status changes to 'yes' and finally wait until the Moving status changes to 'no'. The amount of these elementary operations (10-50) makes the code huge and extremely complicated.

Second issue is the problem of synchronization. Auto-focus is used as a SubVI while collecting images. That means that it is necessary to give this SubVI the control of the motor. That makes it necessary to perform synchronization between VI and SubVI so that each of the motors would wait for another.

Finally, if a programmer wants to change anything in elementary code he/she would need to make changes in plenty of locations.

Therefore, we finally chose to implement all hardware communication into a separate VI which is executed as a new thread. If a LabView program needs to use motors it calls for these motor engine VIs and makes them stop after using.

Motor engine SubVIs

This technology is implemented as two VIs, one for each of the motors (*table_engine.vi*; *camera_engine.vi*.) 'Table' motor engine takes COM port numbers of a motor and of Omron. 'Camera' engine takes link to the camera as well.

The communication between the engines and the main program is performed via global variables. They are stored in a separate VI (*Global_Engine.vi*). Therefore, to set the values of all the variables in a main program a user will need only global variables. There are three types of global variables that we used:

1. Info variables like Position To Go, Actual Position, Direction etc. They define the information written and read from the device.
2. Stop variable
   When the main program changes the status of this variable the loop terminates and closes the session.
3. Special variable which is called 'Table0' and 'Camera0' for two engines accordingly.

   It describes the state of the engine and allows a user to easily launch the required operation. It is an integer variable which can take values:

   1) 0 – engine is free and ready to work
   2) 10-19 – engine is occupied. Motor state is 'Moving'
   3) 20-29 – engine state has just been changed by the program. Motor state has not switched to 'Moving' yet.

   Each operation in an engine has its number from 0 to 9.

To launch an operation:

1) set all necessary info variables to appropriate values
2) change the value of 'Table0' or 'Camera0' variable to '2X' where X is the operation code
3) wait until the value of 'Table0' or 'Camera0' changes to '0'

For example, to shift the cavity to position '100' we must first change 'Pos To Go' variable to the value '100', set the speed, change the 'Table0' value to '21' and wait until it is back to '0'.

Codes of the operations are described below.

Six operations there are elementary: ‘Go to Position’, ‘Step’, ‘Set to Zero’ for both controls. They have codes from 1 to 6. At the same time, the idea of executing SubVIs (which requires the use of motors) in the same way proved to be good. These operations are more complicated – they are executed with separate SubVIs.

For example, to execute auto-focus a user needs to change ‘Camera0’ value to ‘20’ and wait until it is back to ‘0’ which will take from 15 to 30 sec.

To sum up,

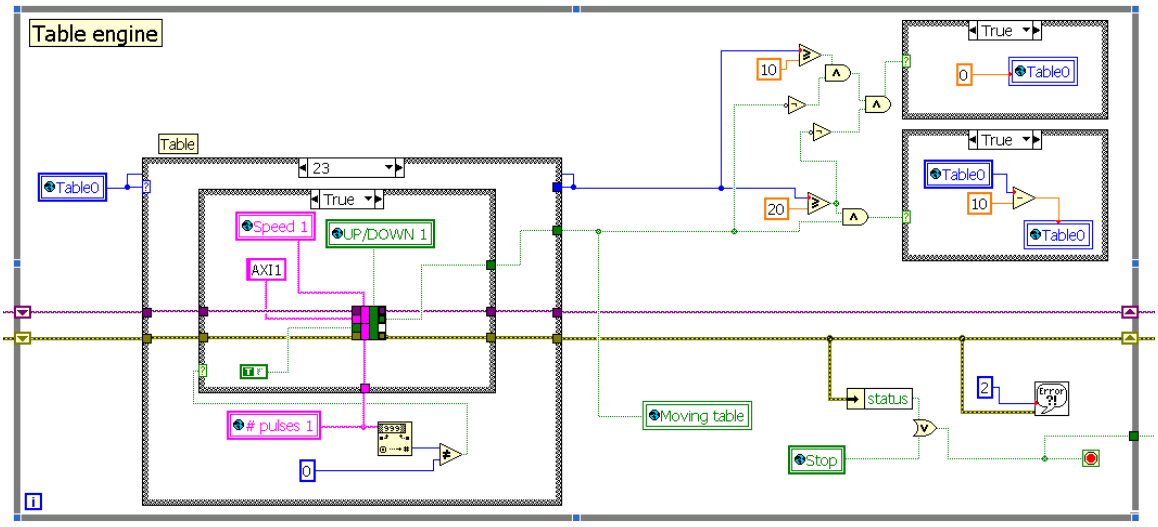
1) Any operation which requires the use of motor is executed via engine SubVIs. An engine must be launched before executing any operation.

2) To execute an operation set all info global variables to appropriate values, then change ‘Table0’ or ‘Camera0’ variables to the value ‘2X’, where X is the operation code. Finally, wait until ‘Table0’ or ‘Camera0’ is back to zero value.

3) Complex operations are executed in a separate SubVI with the simplified engines. A user does not need to control it in any way.

Modification of engines

It is possible that a user would need to add an operation or modify an existing one. In this case, it is necessary to change engines code.

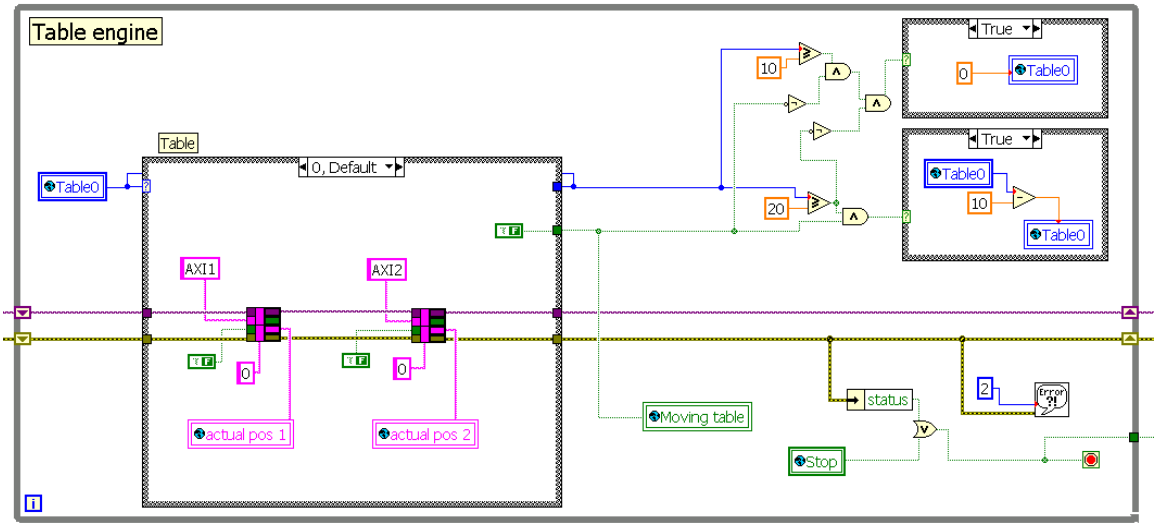


The engines work as loops, continuously checking for updates of variables. The main variables are ‘Table0’ and ‘Camera0’ in two engines accordingly. Different values correspond to different cases in a case structure.

If ‘Table0’ has ‘0’ value nothing is changed. The engine only reads the actual position of two controls.

If ‘Table0’ takes the value ‘2X’ that means that a user launched an operation. The necessary case of case structure is executed and information is written to a serial port. Now the engine starts waiting for ‘Moving’ state from the hardware. It executes the ‘2X’ case until ‘Moving’ switches to ‘yes’ when ‘Table0’ changes to ‘1X’.

If ‘Table0’ is ‘1X’ the engine is waiting for hardware to stop working. As soon as ‘Moving’ state is ‘no’, ‘Table0’ changes to its default ‘0’ value and the engine is ready to be used again.

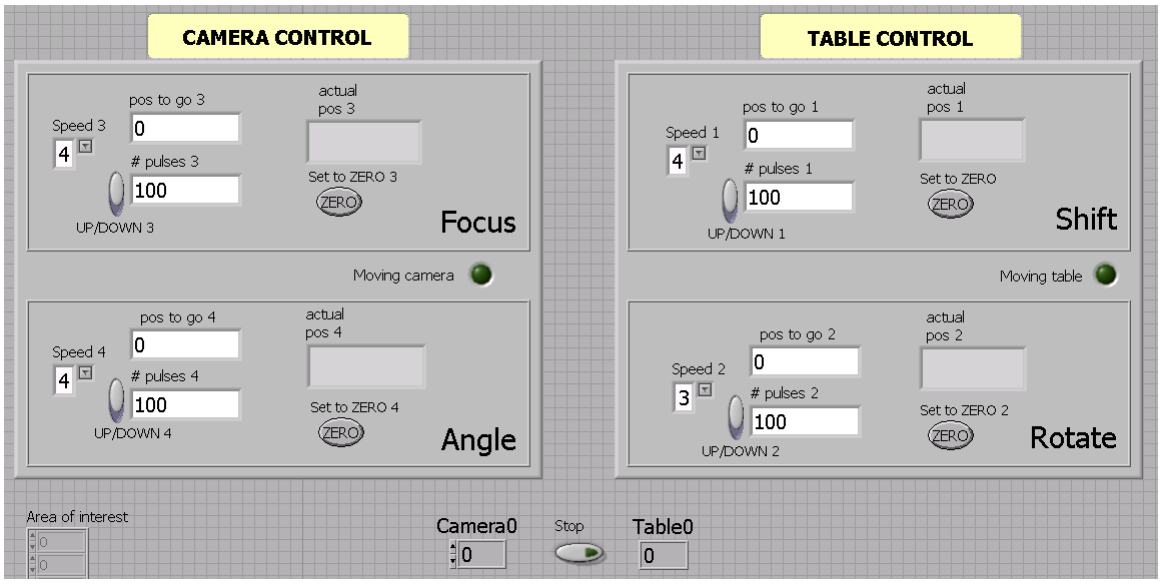


For complex operations all motor manipulations are executed inside a separate SubVI. This is why ‘Working’ state is programmatically set to ‘yes’ as soon as ‘Table0’ value is ‘2X’. When it is ‘1X’ the actual SubVI executing is performed. In this case the loop stops executing and sticks in ‘1X’ case until ‘1X’ case finally changes ‘Working’ value to ‘no’.

Therefore, to add an operation a user must add two cases: ‘1X’ and ‘2X’ where ‘X’ is the first free operation number (0≤X<10, non-integer is OK). For the ‘2X’ case he/she must only set ‘yes’ as an output for ‘Moving’ variable.

For ‘1X’ case a user can insert a code which in most cases would be a SubVI with necessary inputs. Set ‘no’ as output for ‘Moving’ variable.

If necessary a user can also add global variables to global_engine.vi file.



Operations description

Here the operations for ‘Table’ engine and ‘Camera’ engine are described.

Table:

1 - go to position for Shift motor

2 - go to position for Rotate motor

3 - jump steps for Shift motor

4 - jump steps for Rotate motor

5 - set to zero for Shift motor

6 - set to zero for Rotate motor

7 - (recommended instead of #3) precise turn to a required position using Omron indicator.
The shift effect is ‘Table motor’ > ‘Rotate’ is too large. The SubVI for #7 uses Omron indicator to adjust the angle precisely. Make sure that Omron is well calibrated before use.

Camera:

0 - autofocus using given "Area of Interest" as an input

1 - go to position for Focus motor

2 - go to position for Camera Angle motor

3 - jump steps for Focus motor

4 - jump steps for Camera Angle motor

5 - set to zero for Focus motor

6 - set to zero for Camera Angle motor

7 - getting "Area of Interest" from camera

8 - autofocus without using "Area of Interest"

9 - saving image to a file

Here “Area of interest” is a parameter for auto-focus. It is described below.

Autofocus software

Necessity and requirements

For any automatic system of optical inspection, auto-focus system is required. A user or a program must be able to focus on the image by launching only one SubVI. For this reason we developed autofocus mechanism which can perform auto-focusing in 15-30 seconds.

Two images that cannot be told to be different by human in our case correspond to focal length difference of ≈0.1 mm (20 steps of 'Focus' motor). This is the accuracy necessary to be achieved. We achieved the accuracy of 10 steps.

There is a common problem of poor quality of auto-focusing when operating far from focus position. This is because the focus metrics does not change much far away from auto-focus values. Therefore, a user must manually roughly (±500 steps) reach focus position before using auto-focus software.

Focus function

For auto-focus a metrics that shows if an image is in focus is used. There are about 20 different metrics [1]. Best three of them were approbated in our case:

1) absolute gradient

$$F_{th\_grad} = \sum_{Height} \sum_{Width} \left| i(x+1, y) - i(x, y) \right|$$

2) Brenner gradient

$$F_{Brenner} = \sum_{Height} \sum_{Width} \left(i(x+2, y) - i(x, y) \right)^2$$

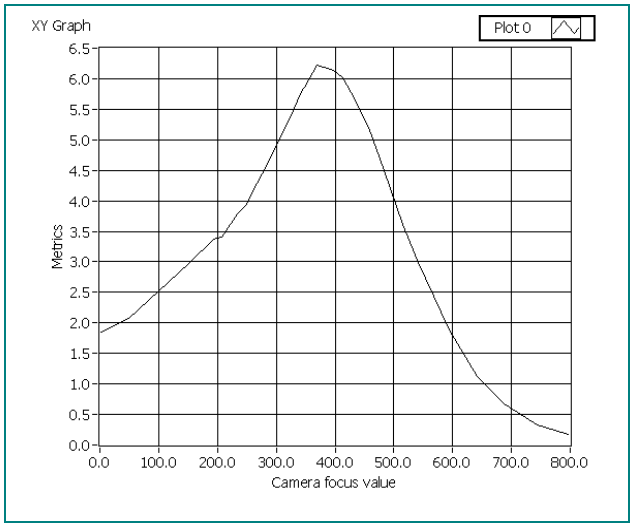
3) variance

$$F_{normed\_variance} = \frac{1}{H \cdot W \cdot \mu} \sum_{Height} \sum_{Width} \left(i(x, y) - \mu \right)^2$$

μ - is mean intensity

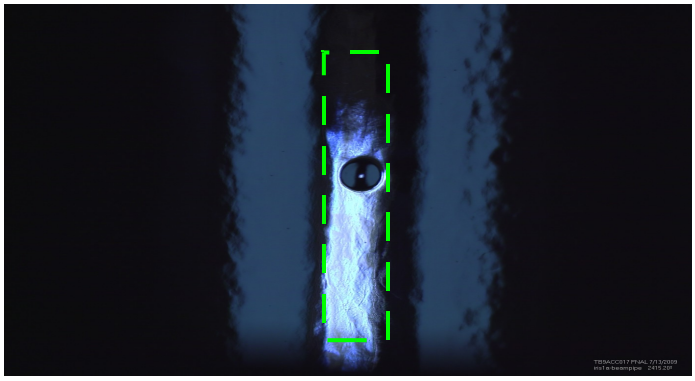
Choice of function and parameters

A focus function is a dependency of this metrics on focal length, which is determined by the camera 'Focus' motor. Absolute gradient showed the sharpest maximum of focus function and therefore was finally chosen as the metrics.



We determined several parameters for this metrics:

Area of interest



It is a part of the image to which the metrics is applied. In our case the image is a bright stripe. The upper and lower borders are 20% and 80% of image height. Several approaches have been developed to determine horizontal borders.

1) Percent of the brightness threshold. The left and right borders limit the area with pixels of more than X% of maximum brightness. In our case X = 40% gives the sharpest peak.

   (Actually the algorithm makes this limitation for three horizontal 1-pixel lines and chooses the shortest interval on x-axis. For each line 5 pixels with brightness more than X% of maximum can be outside the borders.)

2) Overall brightness percent. On equator a picture with two stripes is possible. We developed this approach to choose only one of them to focus on. A shortest x-axis interval which contains X% of sum brightness (overall image brightness) is the necessary area of interest. In our case X = 50% gives the sharpest peak.

   (Actually this shortest x-axis interval is chosen for each of three horizontal 1-pixel lines. The combination of intervals for three lines is chosen as the area of interest.)

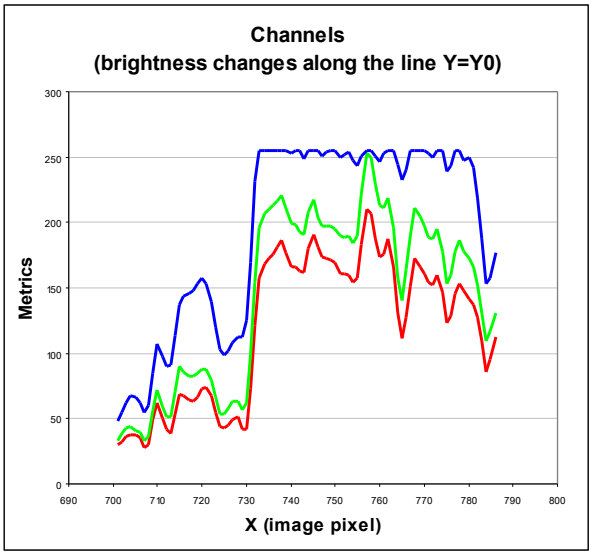
3) Normalized gradient. The approaches above select the brightest area. However, side lights may make a dark area more interesting for us. This approach selects a part with most details in it.

   For ten vertical 10-pixel stripes the Absolute gradient (focus metrics) is calculated. Absolute gradient is normalized by dividing by square root of sum brightness. The 'best' stripes limit the area of interest.

   This approach was not implemented into software.

Channel

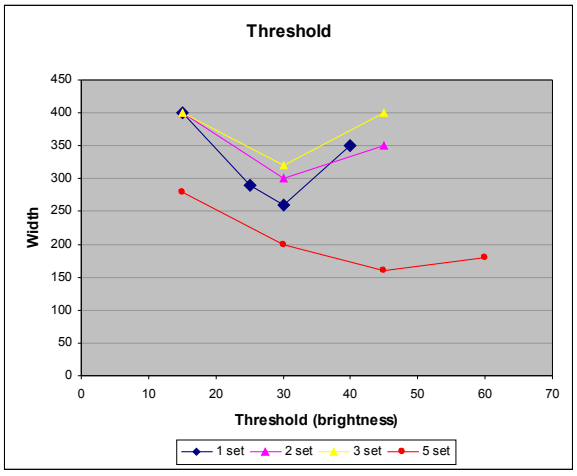
We can choose brightness or either of three color channels to apply focus metrics to. In our case red channel was chosen.



Threshold

Absolute gradient metrics has a threshold parameter which determines the lower limit of the brightness of the pixels used in the metrics.

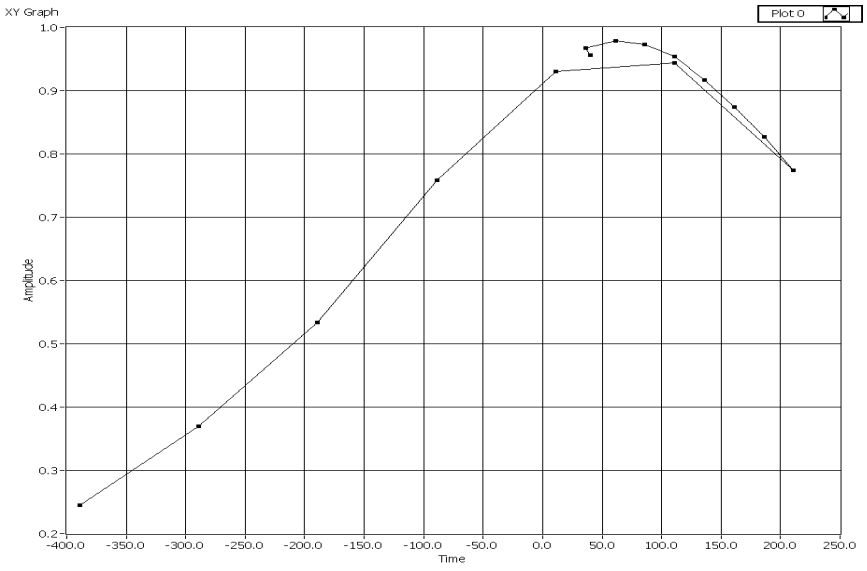
The sharpest peak of focus function corresponds to Threshold = 30 (out of 256).



Approaching focused image

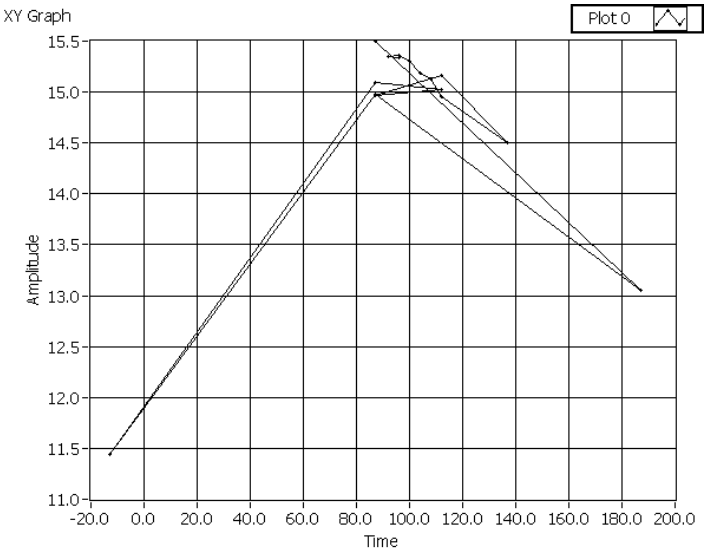
To perform auto-focusing, computer must approach the peak of focus function changing the 'Focus' control of the camera. As we do not know the interval the peak belongs to, the simplest approach to reach the peak is 'Mountain climbing' [2].

The algorithm changes the focal length in certain steps until it finds the maximum. Then it reduces the step length and repeats 'climbing'. We used three step lengths (100, 40, 10).



The precision limits often depend on shift effects.

To reduce shift effect caused by changing directions the special technique is used: every step finishes with motor moving in only 'up' direction. If we need to make step '-10' we perform step '-13' and then '+3'. Therefore, the final position is always after motor steps upwards.



Autofocus software

Auto-focus software is a set of programs in LabView and MatLab. Most LabView programs use engines described above for their operations with motors.

1. *Image_analysis.vi*

   Metrics approbating. LabView program applies a focus metrics defined by a user to a set of several images. Metrics is defined in a special 'Formula node'.

   Within one set images differ by only focus length of the camera. The program makes a plot of focus function depending on focus length.

2. *Auto_focus.vi*

   Performs auto-focus and plots the progress

3. *Imageproc.m*

   MatLab program explores different choices of Area of Interest. Used for performing research on this issue before implementing any algorithm into LabView software. It uses different methods of analysis of an image in order to determine what area is the most informative for our purpose.

4. *Autofocus (SubVI)-[specification].vi*

   Sub-functions perform autofocus and use links to hardware variables as inputs. Used in the 'Camera engine'.

   Autofocus (SubVI)-percent-of-brightness.v and
   Auto-focus-(SubVI)-percent of sum.vi

   use different algorithm to choose area of interest (1st and 2nd accordingly).

   Auto-focus-(SubVI)-Area of Interest as input.vi

   takes Area of Interest as an input.

5. *Getting area of interest (SubVI).vi*

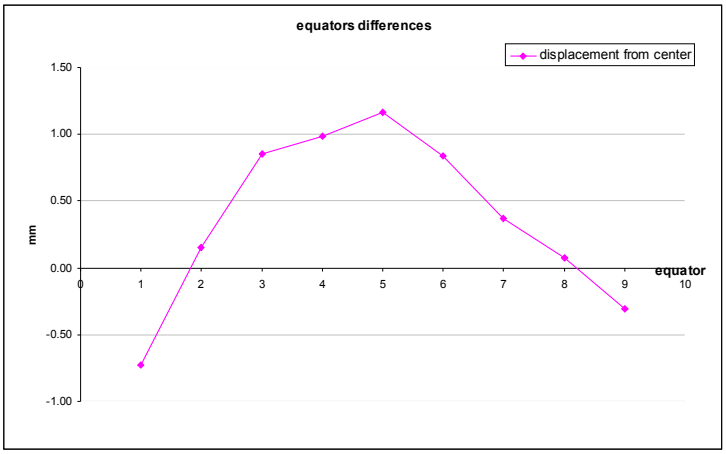
Sub-function calculates the Area of Interest. A user defines the algorithm using special Formula Node. Used in *Auto-focus-(SubVI)-Area of Interest as input.vi*

Software for cavity shape analysis

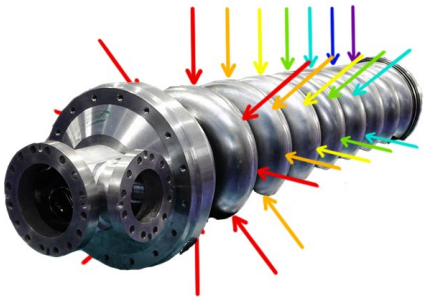
Sample measurements. Results

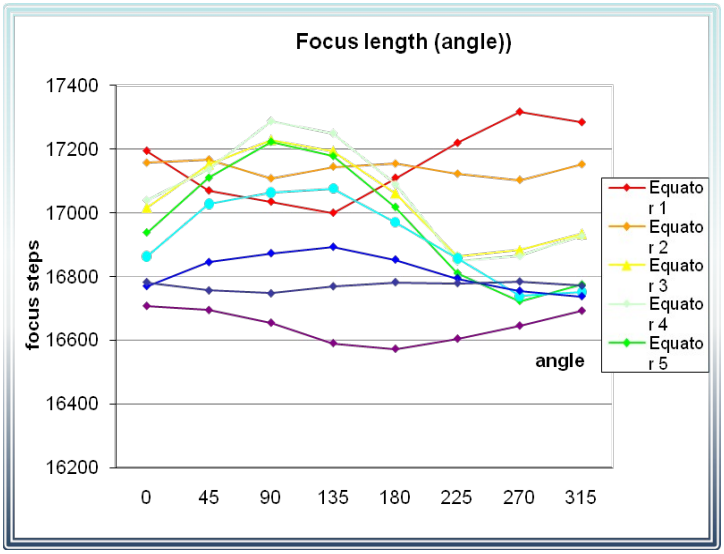
After sample measurements of focal lengths for different angles and different cells we found out that this data can be used to explore the shape of the cavity. Different focus lengths mean different distance between the cylinder with the camera and the cavity.

We created software that takes data, performs analysis and brings the actual cavity shape as an output. Shape variations are about 2 mm in the cavity we studied. The plot below exhibits cell center displacement from the horizontal line. Therefore, the line actually means the shape of the cavity. (Nine x-axis values correspond to nine cells.)



We took focus lengths for 12 equidistant angle values for each of 9 equators and for each of 8 inter-cell welds.





Focus sampling-iris.vi collects all the necessary data from the inter-cell welds. *Focus sampling-equator.vi* collects data from equators. Before using any of two programs make sure the initial 'Table' position is at the first iris or equator and at the zero angle and perform rough focusing on the image.

Both programs execute data collection from every of the cells one by one. The data from every cell is saved into file

D:\Focusing_samples\Iris\X.txt or
D:\Focusing_samples\Equator\X.txt

where X is from 1 to 9.

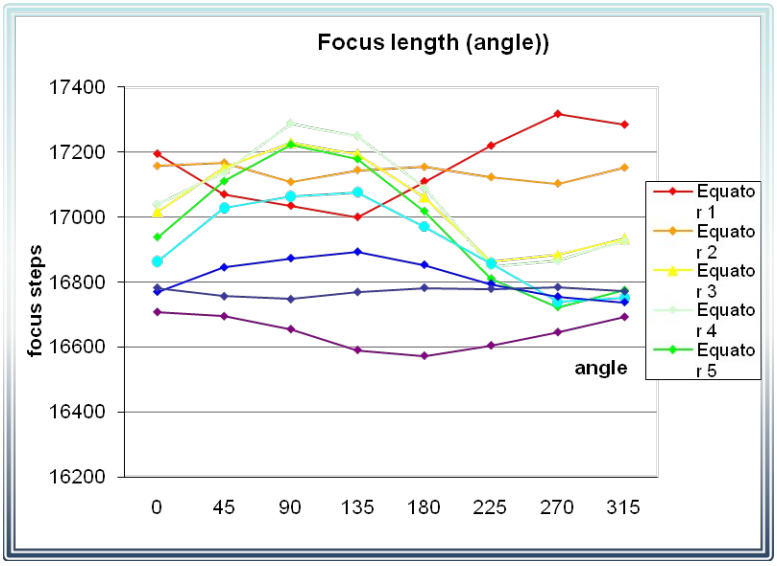
You can choose whether you want to use all the cells or only several of them. If you execute a program again the new data will be added to the existing data in txt files. Therefore, you may want to run a program as many times as necessary to reduce the effect of fluctuations. Indeed the more data is written in the file, the more statistics is used when analyzing the cavity shape.

Due to cell peculiarities (multiple stripes, few details) some cells demonstrate more results variation than the others. That means that for some cells you may want to have more measurements than for the others.

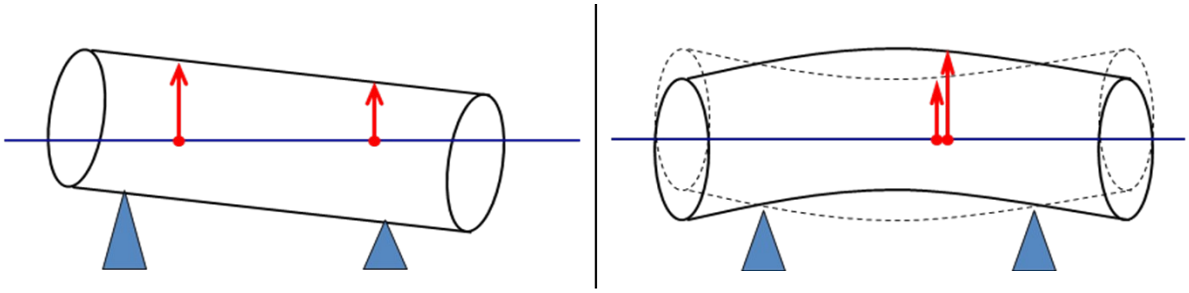
If you want to start a new cavity analysis delete all the files or the folder.

Processing results

The first results we obtained by measuring focus lengths for different angles exhibited 8 harmonics – one per cell. These harmonics differ by offset and by amplitude.



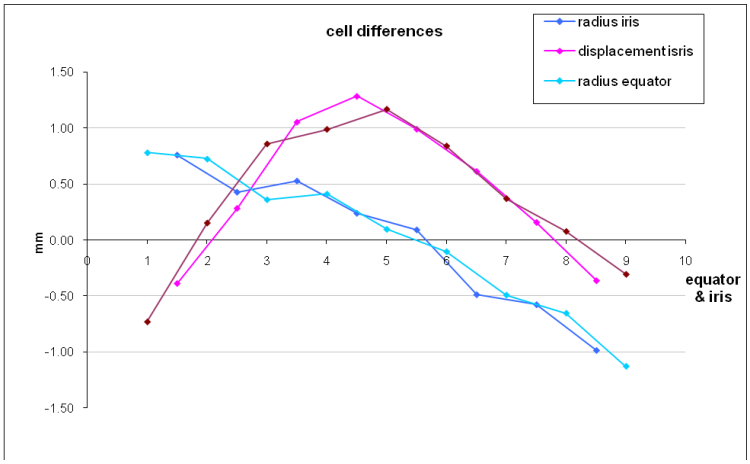
As long as a larger value offset means larger distance from the cell center, the first suggestion was the difference in iris radius. However, different offset accounts for a systematic error:



The cavity is not adjusted parallel to the table level. That means that when it is shifted the position of the cylinder with the camera moves towards the upper surface of the cavity. Thus the focal length which is measured from the cavity level to the upper surface changes from cell to cell. This also illustrates that adjusting the cylinder precisely in the middle is not necessary.

At the same time, the difference in first harmonic amplitude accounts for the bent shape of the cavity. When the cavity rotates the distance to the upper surface changes with one period per 360 degrees of cavity rotation, Moreover, the magnitude of this change is different for different cells. The equators #2 and #8 which lie on the rollers exhibit no focal length change but the cells in the middle have the highest amplitude of the first harmonics.

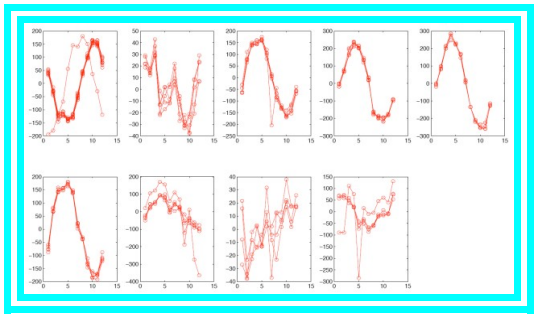
We performed these measurements for irises and equators alike and obtained similar results:



The MatLab program *reader.m* is used to process results.

As an input it takes txt files created by LabView software, that are described above. The result is a plot of displacement of each cell from the horizontal line.

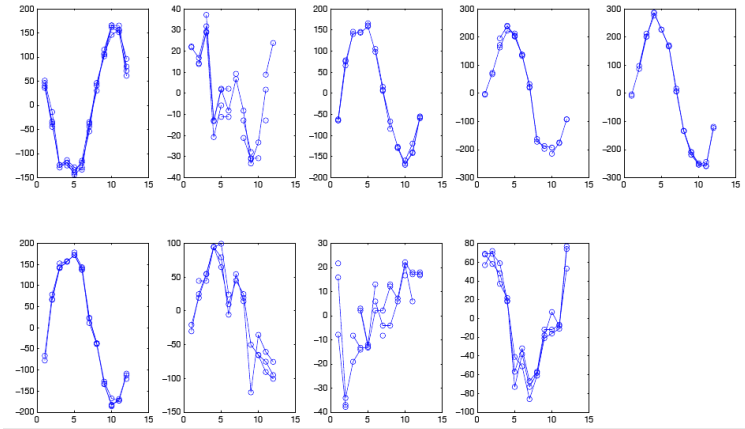
On its way to process the initial data the program performs several steps:



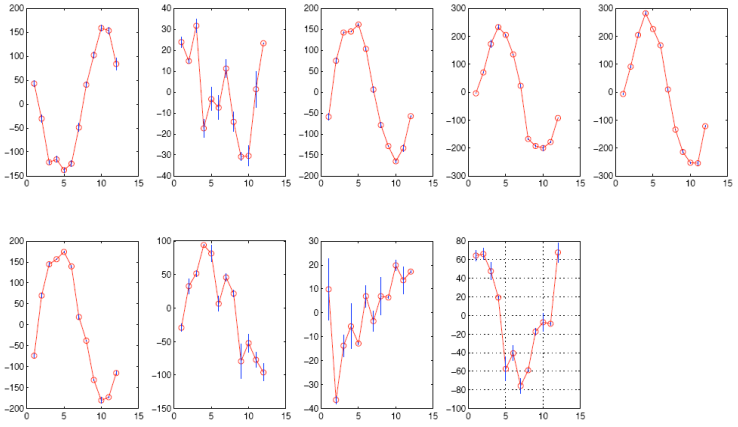
1) The initial information in files is taken from ‘Focus’ camera motor control. Sometimes ‘Focus’ camera motor value is reset to zero. That means that measurement sets written to files in different times can be different by nonzero constant.

   That is why if necessary we add a constant to a measurement set and thus all the focus values for each angle become roughly equal.

2) Now that it is possible to get statistics for each angle, we eliminate outliers. Outlier are defined here as values which differ from the mean angle by more than a standard variation for this angle.

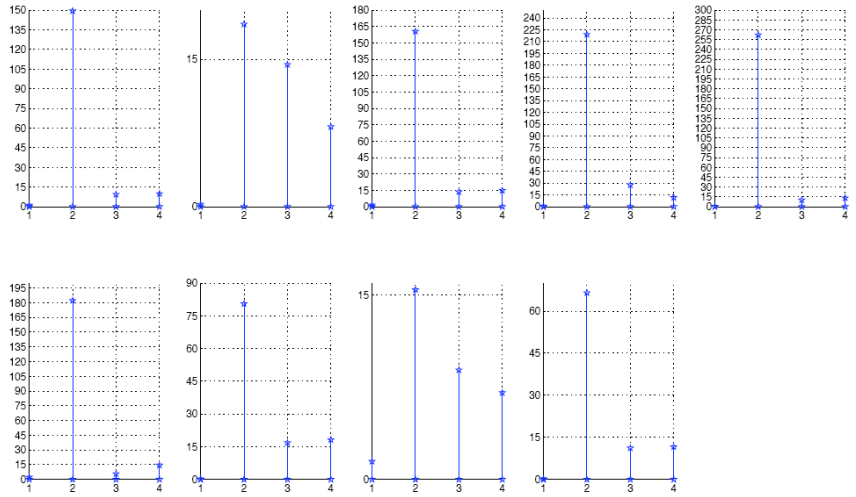


3) For each cell for each angle we transform a set of values into a mean value and a statistical error.

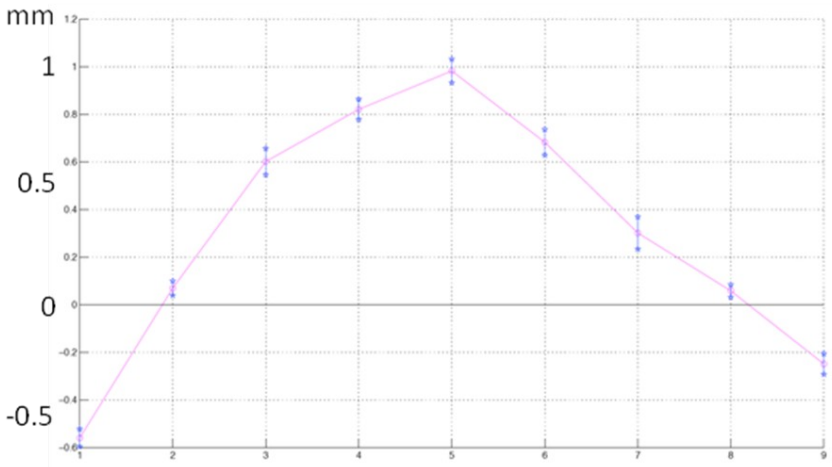


4) Using the mean values for different angles we perform harmonic analysis for every cell. The result of a harmonic analysis are:

   - zero harmonic (which always equals zero because of step 1);
   - first harmonic that accounts for the cell center displacement;
   - second harmonic which corresponds to the ellipse shape of every cell;
   - the residual.



The analysis performed for one of the cavities showed that the cavity center displacement reaches 2 mm. The results are accurate enough to study the cavity shape. At the same time, the accuracy does not allow us to estimate the ellipse shape of every cell (second harmonic value is roughly equal to the residual in most cases.)



Errors and limitations

The accuracy limit of auto-focus is ~10 steps (~0.04 mm). This is also the regular statistical error of cavity shape recognition which can be reduced only by improving auto-focus metrics.

Another kind of error is systematic error. It accounts for differences in areas of interest to focus on. Stripe picture changes from angle to angle and thus areas on different distance from the center are examined.

To reduce this effect the algorithm of choosing part of an image to focus on has been improved. Yet this error cannot be eliminated completely.

The accuracy limit of this technique is illustrated by the value of residual in the harmonic analysis – it is of the order of magnitude of the 2^{nd} harmonics.

Software for image collecting

Algorithm

Image collecting is the ultimate goal. The task is to collect the focused images for the whole weld from all the cells.

Here we will consider only inter-cell iris welds. The software is the same for equator welds with the only difference in the number of cells – 9 for equator instead of 8 for inter-cell welds.

As the number of snapshots to be taken for one cavity is ~1000 the time for auto-focus is the key indicator of efficiency. However, necessity to focus with the precision of only 0.1 mm enabled us to perform auto-focus only about 100 times (12 times per cell). We used interpolation function to work out the focal lengths for all intermediate values.

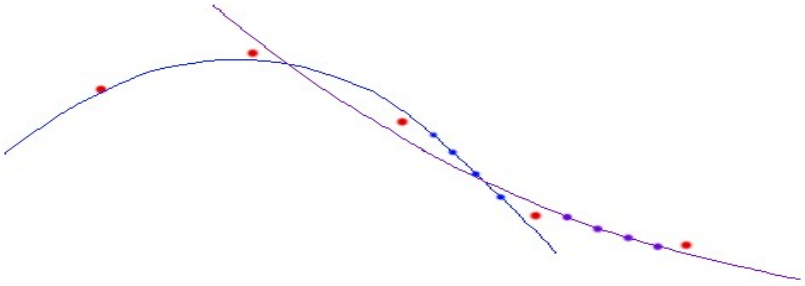
The algorithm includes three steps:

1) moving 30 degrees and performing auto-focus;
2) making interpolation using 4 auto-focus results
3) going 30 degrees back;
   rotating on small angles forward and focusing on images using interpolation results. Saving images. Moving to step 1)
4) shifting to the next cell

Interpolation

Four equidistant values are used for interpolation. The interpolation function is the 2-power polynomial function (parabola). Thus the number of values is one more than the essential number of values for fitting. Interpolation is made for intermediate values between the 3rd and the 4th points.

After getting the next value another interpolation is made.



Conclusion

During the internship of the summer 2010 we have been working on the automation of the optical analysis of the inner surface of SRF cavities. The auto-focus technique has been chosen and implemented; the image collection software has been developed. Besides, further research showed that cavity shape imprecision can be estimated with 10% accuracy. The necessary software has also been created.

At the same, one of the most important results is perhaps development of the software that can be further used for automated optical systems development.

References:

1. Autofocusing algorithm selection in computer microscopy, Yu Sun, S. Duthaler, B.J. Nelson, 2005
2. A New Sampling Method of Auto Focus for Voice Coil, Wei Hsu and Chiou-Shann Fuh